Algorithm Homework #1

경기대학교 컴퓨터공학부

201511837 이상민

1. 아래 문제에 대한 recursive algorithm을 설계하고 시간복잡도를 분석하세요.
(반복문 사용 금지, 재귀호출만 사용하기)

배열에서 최솟값 찾기

```
n = [13,6,9,8,12]

def min(array, com):
    if not array:                      #array의 비교할 것이 남아있지 않다면,
        return com                     #com을 넘기면서 프로그램을 종료한다.
    now = array[0]                     #배열의 첫번째를 now로 선언.
    if now < com:                      #now가 com보다 작다면,
        return min(array[1:], now)     #배열의 첫번째 수가 비교하는 수보다 작으면, now를 com로 대체한다.
    else:                              #now가 com보다 작지 않다면,
        return min(array[1:], com)     #배열의 첫번째가 비교하는 수보다 작지 않다면, com을 유지한다.

print(min(n, n[0]))
```

해당 함수의 시간복잡도는 배열의 길이의 비례만큼 실행되기 때문에 시간복잡도는 O(n)이다.

배열의 원소의 총합 계산하기

```
n = [13,6,9,8,12]

def total(array, sum=0):
    if not array:                  #array의 비교할 것이 남아있지 않다면,
        return sum                 #sum을 넘기면서 프로그램을 종료한다.
    sum += array[0]                #sum에 첫번째 원소를 더한다.
    return total(array[1:], sum)   #이미 더한 원소를 빼기 위해서 0번째 원소를 뺀 배열과 더해져 있는 sum을 리턴한다.

print(total(n))
```

해당 함수의 시간복잡도는 배열의 길이의 비례만큼 실행되기 때문에 시간복잡도는 O(n)이다.

Selection Sort

```
n = [13,6,9,8,12]

def min(array, com):
    if not array:                          #array의 비교할 것이 남아있지 않다면,
        return com                         #com을 넘기면서 프로그램을 종료한다.
    now = array[0]                         #배열의 첫번째를 now로 선언.
    if now < com:                          #now가 com보다 작다면,
        return min(array[1:], now)         #배열의 첫번째수가 비교하는 수보다 작으면, now를 com로 대체한다.
    else:                                  #now가 com보다 작지 않다면,
        return min(array[1:], com)         #배열의 첫번째가 비교하는 수보다 작지 않다면, com을 유지한다.

def selection_sort(array, count=0):
    if count == len(array):                #array의 길이 만큼, count가 진행되었다면,
        return array                       #array를 리턴.

    mini = min(array[count:], array[count])    #min함수를 사용하여서 배열의 가장 최솟값을 찾아 mini에 저장
    index = array.index(mini)              #최솟값 mini의 index번호를 찾는다.
    array[index] = array[count]
    array[count] = mini                    #단계(count)의 맞게 찾은 최솟값을 대상의 array와 위치를 바꾼다.
    count += 1                             #다음 최소값을 찾기 위해서, count를 1증가 시켜 다음 단계를 진행.
    return selection_sort(array, count)

print(selection_sort(n))
```

데이터의 개수가 n개라고 했을 때,

첫 번째 회전에서의 비교횟수 : 1 ~ (n-1) => n-1

두 번째 회전에서의 비교횟수 : 2 ~ (n-1) => n-2

...

(n-1) + (n-2) + ... + 2 + 1 => n(n-1)/2 이므로

선택 정렬의 시간복잡도는 O(n^2)이다.

2. Pancake Sorting

Exercise 9(a), (b) in Chapter 1 of [E] (page 49), (c)는 옵션

9.(a)

```
n = [56,324,23,24,54,83,34]

def filp(array, count=0):
    l = len(array)                         #배열의 길이
    if count == l:                         #배열의 길이와 count 의 횟수가 같아진다면, 종료
        return array

    maxi = max(array[:l-count])            #배열의 0~(l-count)까지 최대값
    index = array.index(maxi)              #최대값의 배열에서의 위치.
    array = array[0:index+1][::-1] + array[index+1:]    #찾은 maxi 를 배열의 첫번째로 위치하게 하고,
    array = list(reversed(array[0:l-count])) + array[l-count:]    #그리고 찾은 maxi 가 l-count 의 위치로 전체 배열을 역전한다.
    count += 1                             #단계를 구분하여 진행하기 위해 count 에 1 증가

    return filp(array, count)              #count 가 배열의 길이와 같아질 때까지 반복

print(filp(n))
```

해당 함수의 시간복잡도는 1~n 까지 탐색해서 가장 큰 팬케이크를 찾아 제일 위로 올리고 뒤집은 다음 1~n-1 까지에서 위와 같은 과정을 반복하기 때문에 2 번의 과정이 계속 반복해서 진행되기 때문에 시간복잡도는 O(n)이다.

최악의 경우는(n 번째가 가장 하단)
n 번째에 가장 큰 수를 놓기 위해 2 회 뒤집기 (총 2 회)
n-1 번째에 그 다음 큰 수를 놓기 위해 2 회 뒤집기 (총 4 회)
…
n-k 번째에 그 다음 큰 수를 놓기 위해 2 회 뒤집기 (총 2k+2 회)
…
4 번째에 그 다음 큰 수를 놓기 위해 2 회 뒤집기 (총 2n-6 회)
3 번째에 그 다음 큰 수를 놓기 위해 2 회 뒤집기 (총 2n-4 회)
3 번째까지 정렬되었고 1,2 번째를 정렬하는 경우 무조건 뒤집는다고 하면(최악의 경우이기 때문에)
최악의 경우의 수는 2n-4 + 1(1,2 번을 무조건 바꾼다는 가정) = 2n-3

9.(b)
베스트 케이스의 경우, 이미 팬케이크가 정렬이 되어있고, 뒤집기가 필요하지 않은 경우이기 때문에 시간복잡도는 O(n)이다.

9.(c)
최악의 경우는(n 번째가 가장 하단)
n 번째에 가장 큰 수를 놓기 위해 3 회 뒤집기 (총 3 회)
n-1 번째에 그 다음 큰 수를 놓기 위해 3 회 뒤집기 (총 6 회)
…
n-k 번째에 그 다음 큰 수를 놓기 위해 3 회 뒤집기 (총 3k+3 회)
…
4 번째에 그 다음 큰 수를 놓기 위해 3 회 뒤집기 (총 3n-9 회)
3 번째에 그 다음 큰 수를 놓기 위해 3 회 뒤집기 (총 3n-6 회)
마지막에 1,2 번째의 윗면이 모두 탄 경우가 최악의 경우이기 때문에 이를 뒤집기 위해서 4 번의 과정이 필요하므로 3n-6+4 = 3n-2 가 최악의 경우가 되고, 위와 과정으로 인해서 시간복잡도는 O(n)이다.